ИСП РАН

# Обфусцирующий компилятор на базе LLVM

**Курмангалеев Шамиль**

**kursh@ispras.ru**

# Задачи обфускации

- Защита от восстановления используемых алгоритмов и структур данных;
- Запутывание вирусов;
- Сокрытие закладок в коде;
- Затруднение генерации эксплоитов на основе анализа патчей, закрывающих уязвимости;
- Препятствие эксплуатации известной уязвимости в коде программ для разных клиентов;
- Простановка «водяных» знаков на версиях программ для разных клиентов;
- Защита от вмешательства в работу программы;
- Затруднение идентификации используемых компонентов с открытым исходным кодом;
- Усложнение идентификации автора кода.

# Подход к реализации

- Многие алгоритмы обфускации требуют наличия информации характерной для компиляторов
- Встраивание защиты во время компиляции позволяет увеличить ее стойкость и скорость разработки защиты
- Во время компиляции мы обладаем максимальной информацией о программе
- Автоматическая поддержка нескольких целевых архитектур
- Желательно сохранить устоявшийся процесс разработки ПО

Требуется компиляторная инфраструктура

# LLVM

**Компиляторная инфраструктура с открытыми исходными кодами**

- **Модульная и расширяемая архитектура**
- **Является статическим компилятором, а так же имеет возможность JIT'тить биткод**

**Поддерживает несколько фронтэндов**

- **C, C++, Objective-C (Clang, GCC/dragonegg)**
- **Ruby (Rubinius, MacRuby)**

**Поддерживает множество целевых архитектур**

- **ARM, Alpha, Intel x86, Microblaze, MIPS, PowerPC, SPARC, …**

**Промежуточное представление играет центральную роль в процессе компиляции (LLVM IR)**

- **Все оптимизации реализованы как компиляторные проходы преобразования "LLVM IR to LLVM IR"**
- **Анализ кода, может быть реализован как отдельный проход, а его результаты могут разделять несколько проходов трансформирующих код**
- **Все машинно-зависимые оптимизации происходят в отдельном бэкэнде для каждой машины**

# Поддерживаемые преобразования

- Перемещение локальных переменных в глобальную область видимости
- Приведение графа потока управления к плоскому виду
- Переплетение нескольких функций в одну
- Сокрытие вызовов функций
- Создание несводимых участков в графе потока управления
- Шифрование константных строк, используемых программой
- Вставка в код фиктивных циклов, из 1 итерации (do-while)

# Поддерживаемые преобразования

- Размножение тел функций
- Разбиение целочисленных констант
- Модификация CFG стандартными средствами компилятора, без цели оптимизации
- Переупорядочивание и добавление локальных переменных

Некоторые из указанных методов используют непрозрачные предикаты

Сборка больших проектов

Пример: Связка LLVM+Clang с запутывающим преобразованием диспетчер (запутывался только код обфускатора).

Время обфускации увеличилось в 1.5 раза, выходные файлы одинаковы

# Существующие решения

**Obfuscator**

(https://github.com/obfuscator-llvm/obfuscator/wiki/Features)

- Преобразования промежуточного представления LLVM
- Доступные опции
  - Вставка избыточных вычислений a = b & c => a = (b ^ ~c) & b
  - Вставка непрозрачных предикатов
  - Преобразование диспетчер (Control Flow Flattening)

# Существующие решения

- **Confuse**: LLVM-based Code Obfuscation (Columbia University)
  - Обфускация строк – замена строк их хэшами
  - Вставка избыточных вычислений
  - Вставка непрозрачных предикатов (основанных на математических тождествах) и переменных
- **Morpher** http://morpher.com/
  - Усложнение графа потока управления (CFG arches meshing)
  - Клонирование базовых блоков
  - Защита констант
  - Клонирование функций
  - Переплетение функций
  - Вставка непрозрачных предикатов
  - Вставка фиктивных циклов

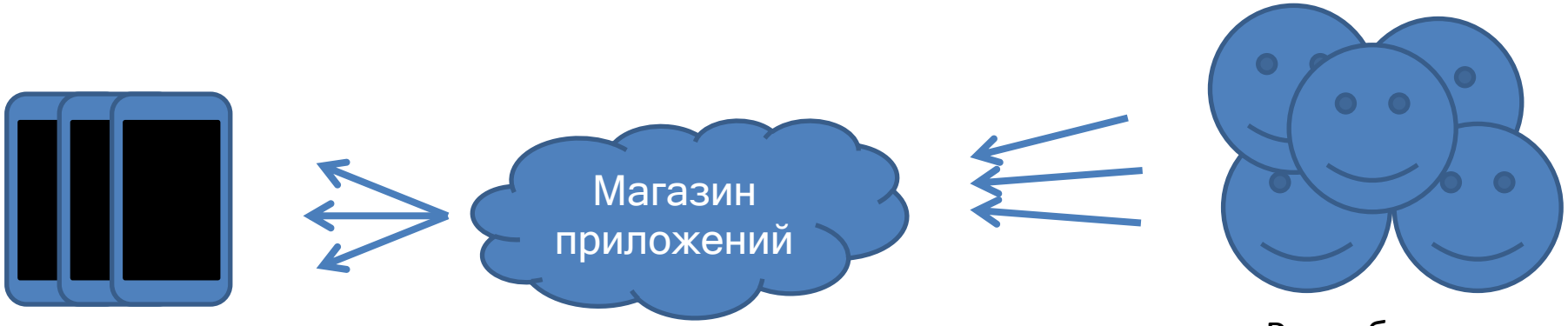
# Существующие решения

- **Tigress** (source-to-source based on CIL infrasructure) – Cristian Collberg (http://tigress.cs.arizona.edu/)
  - Виртуализация функций
  - Диспетчеризация
  - Разбиение функций
  - Переплетение функций
  - Вставка непрозрачных предикатов
  - Генерация дополнительных аргументов функций
  - Замена строкового представления чисел непрозрачными выражениями (“42” => opaque expression)

# Дополнительные применения обфускации

- Препятствие эксплуатации уязвимостей
  - Переполнение буфера – перезапись данных за пределами буфера
- Защита от сохранения страниц памяти приложения на диск (antidump)
  - Требуется сократить время пребывания данных в памяти в открытом виде,  желательно делать это автоматически

# Модель распространения приложений



# Предлагаемые трансформации

Для каждого клиента генерируется уникальный бинарный образ:

- перестановка местами функций в модуле;
- добавление случайного числа локальных переменных в функции;
- переупорядочивание локальных переменных.

- Адреса и смещения в различных экземплярах разные, что затрудняет эксплуатацию известной уязвимости

# Автоматическое шифрование буферов

- Буфер расшифровывается перед каждым обращением.
- Автоматическое шифрование после обращения не всегда возможно (имеются операции с указателями)

```
sub     rcx, rbp
mov     qword ptr [rsp+100h+n], rcx
call    decryptn
mov     rcx, qword ptr [rsp+100h+n]
mov     edx, 64h
mov     rsi, rbp          ; src
mov     rdi, rbx          ; dest
sub     rdx, rcx          ; n
call    _strncat
mov     esi, 64h
mov     rdi, rbx
xor     eax, eax
call    encryptn
```

Пользователь может создать свои функции шифрования/дешифрования:
char *encrypt(char *s);
char *decrypt(char *s);
char *encryptn(char *s, int len);
char *decryptn(char *s, int len);

# Возможные направления работы

- Return-oriented programming  (ROP) основана на идее построения цепочки адресов возврата  на так называемые «гаджеты» ( полезная инструкция; ret).
- Для предотвращения этой атаки требуется перекомпилировать все библиотеки, используемые программой (в т.ч. системные), так чтобы они не содержали «гаджеты»
- Упомянутые техники обфускации  также затрудняют проведение подобной атаки, если они применялись в целях диверсификации.

# Результаты

Тестирование проводилось на модельном примере, содержащем уязвимость переполнения буфера.

Эксплоит успешно отработавший на версии кода доступной атакующему, был неработоспособен на других сборках программы.

Влияние на производительность на программе SQLite замедление составило 30%

ИСП РАН

# Спасибо за внимание

? Вопросы ?